

Квантизация цветов в изображении методом медианного сечения

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной статье описан алгоритм медианного сечения. Реализация алгоритма представлена на языке программирования C. Результатом исследования станет программа с описанием разработки и оценкой сложности методов.

Ключевые слова: Квантизация, растровая графика, обработка изображения.

Quantization of colors in an image by the median section method

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This article describes the median section algorithm. The implementation of the algorithm is presented in the C programming language. The result of the research will be a program describing the development and evaluating the complexity of the methods.

Keywords: Quantization, raster graphics, image processing.

1 Введение

1.1 Актуальность

Квантование цветов в изображении – это процесс уменьшения количества различных цветов, используемых в изображении, с сохранением визуального качества. Квантование позволяет значительно уменьшить размер изображения, что важно для хранения и передачи данных. Некоторые форматы файлов и устройства не поддерживают изображения с высоким разрешением цвета, примером служит формат gif. Также уменьшение количества цветов является важным этапом в алгоритмах сжатия изображений, таких как JPEG и PNG. Оно позволяет эффективно уменьшать объем данных без значительной потери качества. В задачах машинного обучения и компьютерного зрения часто требуется обработка большого количества изображений. Квантование цветов помогает уменьшить сложность данных и ускорить процессы обучения.

1.2 Обзор исследований

М.Т.Нариманлы исследовал уменьшение шума во время квантизации сигналов яркости телевидения [1]. Н.И.Амиантов описал алгоритм сжатия

изображений на основе динамического сжатия палитры [2]. А.Г.Агаджанян изучил задачи и методы тоновой аппроксимации растровых монохромных мультитоновых изображений [3]. А.Р.Мазурина рассмотрела способ уменьшения количества цветов изображения [4]. Ю.А.Маньяков, А.И.Сорокин представили метод сегментации изображений на основе квадродерева [5]. А.И.Сорокин, В.Н.Волков, Ю.О.Сорокина провели исследование адаптивного алгоритма квантования цветового пространства на основе медианного сечения [6].

1.3 Цель исследования

Цель исследования – описать алгоритм квантизации цветов в изображении методом медианного сечения.

2 Материалы и методы

Для реализации используется язык программирования «С». Оценка сложности алгоритма представляется в виде асимптотического поведения функций Big O notation.

3 Результаты и обсуждения

Алгоритм медианного сечения цвета основан на рекурсивном разбиении цветового пространства. Он разделяет цветовое пространство на подпространства с помощью медианного значения, пока не будет достигнуто заданное количество цветов. Алгоритм начинается с инициализации, в ходе которой собираются данные всех пикселей изображения, и формируется список всех цветов. Этот список представляет собой единый куб в трехмерном цветовом пространстве, например, RGB. Для каждого куба определяются минимальные и максимальные значения компонент цвета (R, G, B), чтобы определить его размеры. Затем алгоритм выполняет рекурсивное разбиение кубов. Для этого сначала выбирается ось разбиения, по которой размер куба максимален, чтобы сбалансировать разбиение. Далее пиксели сортируются по значениям выбранной компоненты цвета, и находится медианное значение этой компоненты. Это значение используется для разбиения куба на два новых подпространства. Процесс рекурсивного разбиения повторяется до тех пор, пока не будет достигнуто заданное количество цветов в палитре. После завершения разбиения, каждый из полученных кубов представляется одним цветом, который обычно является средним значением цветов, находящихся в этом подпространстве (рис.1).

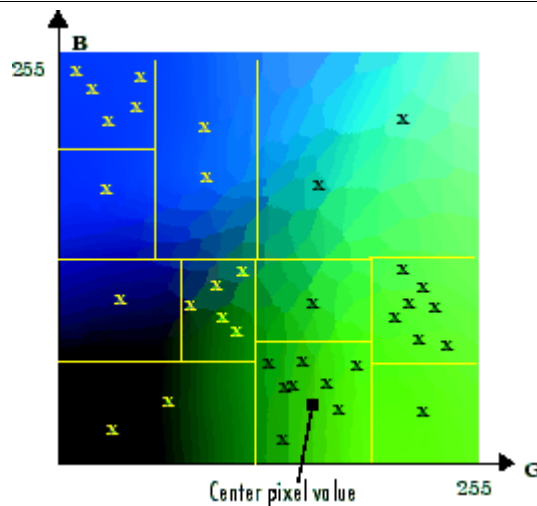


Рис. 1. Двухмерная проекция деления цветов медианным сечением

Для упрощения реализации созданы структуры данных для хранения данных пикселей и палитры цветов (рис.2).

```
typedef struct {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} Color;

typedef struct {
    Color* color;
    uint32_t size;
} Palette;
```

Рис. 2. Структуры хранения цвета и палитры

Сначала нужно подготовить данные, для этого необходимо присвоить значения пикселей в массив цветов. Как видно по содержанию структур, предполагается, что глубина цвета пикселя равна 24 бита. Дополнительно нужно из всех пикселей собрать только уникальные цвета. Сложность алгоритма из-за сортировки $O(n \cdot \log(n))$, где n – количество пикселей (рис.3).

```
uint32_t get_palette(Color *pixels1, uint32_t count_pixels, Palette *unique)
{
    Color* pixels=(Color*)malloc(count_pixels*sizeof(Color));
    memcpy(pixels, pixels1, count_pixels * sizeof(Color));
    qsort(pixels, count_pixels, sizeof(Color), colorcmp);

    unique->color[0]=pixels[0];
    unique->size=1;

    Color tmp=pixels[0];

    for(uint32_t i=0; i<count_pixels;i++)
    {
        if((tmp.red!=pixels[i].red) || (tmp.green!=pixels[i].green) || (tmp.blue!=pixels[i].blue))
        {
            tmp=pixels[i];
            unique->color[unique->size++]=tmp;
        }
    }

    free(pixels);
    return unique->size;
}
```

Рис. 3. Метод получения уникальных цветов изображения

Функция медианного сечения принимает четыре аргумента. К необходимым значениям относятся максимальное количество цветов в палитре, указатель на массив палитры, значение количества пикселей в массиве и указатель на структуру палитры, в которую будут сохранены результаты.

Первым шагом функция проверяет, если размер палитры достиг 1 или количество пикселей равно 1, то это базовый случай рекурсии. В этом случае она вычисляет средний цвет для текущего набора пикселей с помощью функции дополнительного метода, сохраняет этот цвет в палитру и увеличивает размер палитры на 1.

В противном случае функция начинает процесс поиска минимальных и максимальных значений для каждой из цветовых компонент (красной, зеленой и синей) в массиве пикселей. Это делается в цикле, проходящем по всем пикселям. На каждом шаге цикла функция обновляет минимальные и максимальные значения для красной, зеленой и синей компонент.

После нахождения минимальных и максимальных значений компонент функция вычисляет диапазоны для каждой из них: `rRange` для красной, `gRange` для зеленой и `bRange` для синей. Эти диапазоны определяют, какая компонента имеет наибольший разброс значений. Исходя из этого, выбирается компонента с наибольшим диапазоном, и массив пикселей сортируется по значениям этой компоненты с использованием функции `qsort` и соответствующей функции сравнения (`redcmp`, `greencmp` или `bluecmp`).

После сортировки функция находит медиану массива пикселей, которая делит массив на два подмножества с равным (или почти равным) количеством пикселей. Затем функция рекурсивно вызывается для обоих подмножеств, уменьшая значение размера палитры вдвое. Это приводит к дальнейшему разбиению цветового пространства до тех пор, пока не будут достигнуты базовые случаи рекурсии (рис.4).

```

int median_section(int max_palette_size, Color* pixels, int count_pixels, Palette *palette)
{
    if(max_palette_size == 1 || count_pixels == 1)
    {
        palette->color[palette->size] = mean_color(pixels, count_pixels);
        palette->size++;
        return 1;
    }

    int rMin = 255, rMax = 0, gMin = 255, gMax = 0, bMin = 255, bMax = 0;
    for (int i = 0; i < count_pixels; i++) {
        if (pixels[i].red < rMin) rMin = pixels[i].red;
        if (pixels[i].red > rMax) rMax = pixels[i].red;
        if (pixels[i].green < gMin) gMin = pixels[i].green;
        if (pixels[i].green > gMax) gMax = pixels[i].green;
        if (pixels[i].blue < bMin) bMin = pixels[i].blue;
        if (pixels[i].blue > bMax) bMax = pixels[i].blue;
    }

    int rRange = rMax - rMin;
    int gRange = gMax - gMin;
    int bRange = bMax - bMin;

    if (rRange >= gRange && rRange >= bRange)
    {
        qsort(pixels, count_pixels, sizeof(Color), redcmp);
    }
    else if (gRange >= rRange && gRange >= bRange)
    {
        qsort(pixels, count_pixels, sizeof(Color), greencmp);
    }
    else
    {
        qsort(pixels, count_pixels, sizeof(Color), bluecmp);
    }

    //находим самую длинную сторону
    uint32_t median = count_pixels/2;

    /*делаем сечение куба, получаем 2 параллелипипеда с равным (+-1) количеством пикселей
    рекурсивно повторяем*/
    median_section(max_palette_size/2, pixels, median, palette);
    median_section(max_palette_size/2, pixels+median, count_pixels-median, palette);

    return 1;
}

```

Рис. 4. Реализация метода медианного сечения

Метод вычисления среднего цвета тривиален, и не нуждается в описании (рис.5).

```

Color mean_color(Color *pixels, uint32_t count_pixels)
{
    uint32_t r_sum=0;
    uint32_t g_sum=0;
    uint32_t b_sum=0;
    for(int i=0; i<count_pixels; i++)
    {
        r_sum+=pixels[i].red;
        g_sum+=pixels[i].green;
        b_sum+=pixels[i].blue;
    }
    Color mean;
    mean.red=r_sum/count_pixels;
    mean.green=g_sum/count_pixels;
    mean.blue=b_sum/count_pixels;
    return mean;
}

```

Рис. 5. Определение среднего значения цвета

Функция qsort требует реализацию метода сравнения элементов в массиве, реализации методов для каждого компонента цвета представлены на рисунке 6.

```
int redcmp(const void* a, const void* b)
{
    return (*(Color*)a).red - (*(Color*)b).red;
}
int greencmp(const void* a, const void* b)
{
    return (*(Color*)a).green - (*(Color*)b).green;
}
int bluecmp(const void* a, const void* b)
{
    return (*(Color*)a).blue - (*(Color*)b).blue;
}
```

Рис. 6. Реализация функций сравнения

Нахождение минимальных и максимальных значений для каждой из компонент (красный, зеленый, синий) требует $O(n)$ времени, где n - количество уникальных цветов. Сортировка массива пикселей в среднем случае требует $O(n \cdot \log(n))$ времени. В худшем случае рекурсивная функция вызывается для каждого пикселя, что приводит к логарифмической глубине рекурсии. Таким образом, количество вызовов функции для каждой рекурсивной ветви составляет $\log(k)$, где k - количество цветов в палитре. Вычисление среднего значения для каждого бокса требует $O(n)$ времени, где n - количество цветов в кубе.

Для нахождения ближайшего цвета из палитры к заданному пикселю используется взвешенное евклидово расстояние в цветовом пространстве. Этот метод учитывает различную восприимчивость человеческого глаза к разным цветам, применяя различные веса к компонентам красного, зеленого и синего цветов. Веса составляли 30 для красного, 59 для зеленого и 11 для синего цвета. Эти веса подобраны таким образом, чтобы соответствовать чувствительности человеческого глаза, который наиболее чувствителен к зеленому цвету, менее чувствителен к красному и еще менее чувствителен к синему цвету. Это делается для улучшения точности визуального восприятия при сопоставлении цветов, а также ускорению работы алгоритма.

Также добавлена проверка на сравнение следующего пикселя, который в реальных изображениях, очень часто идентичен предыдущему. Дополнительная проверка значительно ускоряет процесс сопоставления цветов (рис.7).

```
int find_closest_color(Color pixel, Palette palette)
{
    int i, fi, best_color, f_min=1000000;
    for(i=0; i<palette.size; i++)
    {
        C++;
        fi=30*(palette.color[i].red - pixel.red) * (palette.color[i].red - pixel.red)+
            59*(palette.color[i].green - pixel.green)*(palette.color[i].green - pixel.green)+
            11*(palette.color[i].blue - pixel.blue)*(palette.color[i].blue - pixel.blue);
        if(fi<f_min)
        {
            best_color=i, f_min=fi;
        }
    }
    return(best_color);
}

void map_pixels_to_palette(Color* pixels, size_t num_pixels, Palette palette, Color* new_pixels) {
    for (size_t i = 0; i < num_pixels; ++i) {
        if(i>1 && colorcmp(pixels+i, pixels+(i-1))==0)
        {
            new_pixels[i]=new_pixels[i-1];
        }
        else
        {
            new_pixels[i] = palette.color[find_closest_color(pixels[i], palette)];
        }
    }
}
```

Рис. 7. Сопоставление оригинального цвета пикселя цвету из палитры

Для представления результата работы взят пример изображения с 285 772 800 пикселями и 239 744 количеством уникальных цветов. В качестве размера полученной палитры было выбрано 256 цветов. Оригинальное и обработанное изображения представлены на рисунках 7,8 соответственно. Далее нужно загрузить все данные в одну функцию. Сравнительный метод соседних пикселей из функции нахождения ближайшего цвета из палитры позволил сократить количество итераций для данного изображения с 285 772 800 до 212 239 872, что составляет почти 26%. Как и следовало ожидать, главной слабостью алгоритма медианного сечения является плавный переход градиента одного цвета. Это отчётливо заметно по ступенчатому переходу оттенков синего цвета неба в результирующем изображении.



Рис. 8. Оригинальное изображение



Рис. 9. Изображение после квантования цветов

В результате исследования был разобран алгоритм квантования цвета медианным сечением. Описан способ реализации на языке С. Продемонстрированы попытки оптимизации и оценки сложности алгоритма.

Библиографический список

1. Нариманлы М.Т. Уменьшение шума во время квантизации сигналов яркости телевидения // В сборнике: Актуальные вопросы подготовки кадров по энергетическим специальностям. II-я республиканская научная конференция. sumqayit, 2021. с. 86-88.
2. Амиантов Н.И. Алгоритм сжатия изображений на основе динамического сжатия палитры // Молодежный научно-технический вестник. 2014. № 12. С. 34.
3. Агаджанян А.Г. Задачи и методы тоновой аппроксимации растровых монохромных мультитоновых изображений // Инженерный вестник Дона. 2018. № 3 (50). С. 74.
4. Мазурина А.Р. Уменьшение количества цветов изображения // В сборнике: Материалы Научной сессии. в 2 т.. Волгоград, 2020. С. 184-188.
5. Маньяков Ю.А., Сорокин А.И. Метод сегментации изображений на основе квадродерева // Системы и средства информатики. 2020. Т. 30. № 4. С. 102-112.
6. Сорокин А.И., Волков В.Н., Сорокина Ю.О. Адаптивный алгоритм квантования цветового пространства на основе медианного сечения // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2018. Т. 8. № 2 (27). С. 63-72.
7. Дюжикова Л.Э., Лонский И.И. Медианная фильтрация изображений // Известия высших учебных заведений. Геодезия и аэрофотосъемка. 1994. № 6. С. 111-129.
8. Волчкова Д.С. Изменение цвета изображения на основе объединения алгоритмов сегментации // Colloquium-Journal. 2018. № 3-1 (14). С. 28-31.